Introduction to Gradle

- Comparing Build Tools
- Gradle and Groovy
- Tasks
- Dependency Management

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Build Tool

- Most Java IDEs, e.g. Eclipse, provide functionality for building projects
- However, using a separate build tool has advantages:
  - Portability - don't need the IDE running to build
  - Automated builds, perhaps on a schedule for **continuous integration**
- Java has several popular build tools available, including:
  - Ant
  - Maven
  - Gradle

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Apache Ant

- **Apache Ant** is a mature build tool in which you write XML build scripts
- Though Ant is well-known and highly supported by IDEs, XML is verbose and not well suited to conditional tasks
- Ant by itself provides no **dependency management** for libraries (Apache Ivy adds that function)



Ant build scripts can become very verbose and difficult to maintain.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Apache Maven

- **Apache Maven** is a modern build tool that includes **dependency management** and supports the notion of **convention over configuration**
- Maven is based on the concepts of a build **lifecycle** composed of **phases**, which in turn are composed of **goals**
- You describe the build using XML (the **project object model (POM)**, which can be verbose



Maven is fairly strict about enforcing a project structure. In addition, extending Maven by writing custom plugins can be troublesome.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introducing Gradle

- **Gradle** is a "next generation" build tool that builds on concepts from Ant and Maven
- You write Gradle scripts using syntax based on the **Groovy** language



The home page for Gradle is at: http://gradle.org.

Gradle is designed to take the best ideas from Ant and Maven and make it easy (fun?) to write build scripts. Since Groovy is based on Java, learning Gradle is straightforward for Java developers and the syntax is more readable and succinct than XML.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Groovy?

- Groovy is a dynamic, typing optional language with syntax similar to Java that runs on the JVM
- Some differences from Java:
  - Semicolons optional
  - Delimit strings with single or double quotes (double quoted support **interpolation**)
  - Can omit parenthesis for method calls
  - **print** and **println** shortcuts for System.out.println



The home page for Groovy is at: http://www.groovy-lang.org/index.html

Like Java 8, Groovy supports functional programming and closures.

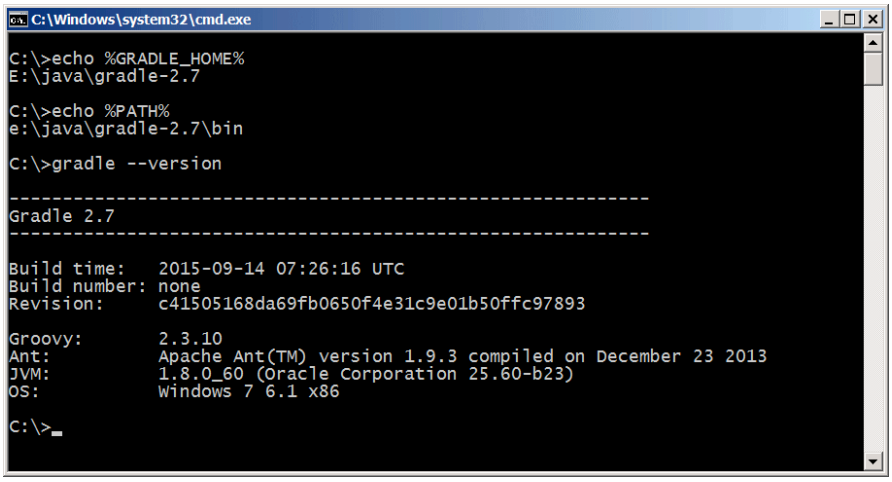
Interpolated strings can contain variable references, for example:

```
def name = "John Adams"
println "Hello, $name"
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Downloading and Installing Gradle

- Installing Gradle is easy - download a ZIP file and update your computer's environment to include GRADLE_HOME and so that PATH includes GRADLE_HOME/bin



Mac OS X users can also install Gradle using the Homebrew package manager.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing Gradle Build Files

- Gradle defines a Groovy **Domain Specific Language** (DSL) that you can use to write scripts
- Build scripts typically contain **configuration blocks** and define and/or use **tasks**
- You can use the overloaded << operator to add actions (instructions) to a task

```
1    // define a task
2    task hello
3
4    // configure a task
5    hello {
6      ext.firstname = 'John'
7    }
8
9    // append action(s) to the task
10   hello << {
11     println "Hello $firstname"
12   }
```


Gradle scripts run through three phases in order: initialization, configuration and execution. The configuration block thus runs before the task's actions.

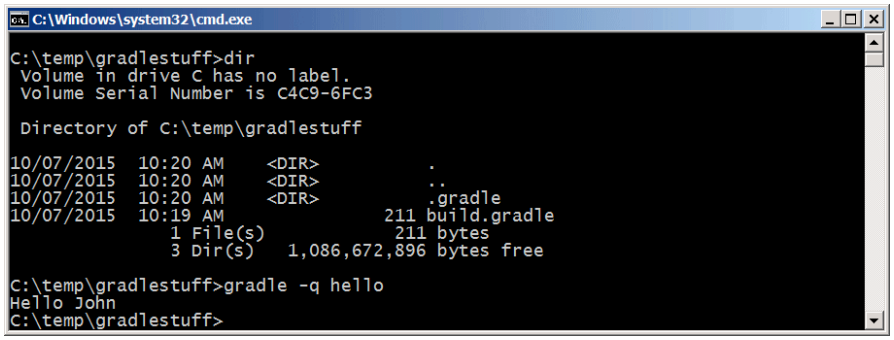
In the script above, we define a task named 'hello'. Tasks are actually Groovy objects that derive from Gradle's DefaultTask type. In the configuration block that follows, we add a so-called "extra" (ext) property. Then in an action for the task, we access the property.

You can always 'drop down' and write actual Groovy code in a build script, but that's discouraged, since it makes the scripts harder to maintain.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Running a Build From the Command Line

- By default, Gradle looks for a file named **build.gradle**
- When you run a script from the command line, you can specify the task as an argument



Gradle has several predefined tasks, including one named 'tasks' that displays a summary of all of the tasks available in the script

You can use the -q flag for 'quiet mode', supressing Gradle diagnostics.

You can use the -b flag to specify the build file name if it's different from build.gradle.

Gradle also provides a GUI mode that you can invoke with the -gui flag.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dependent Tasks

- Task often need to execute in a certain order - for example, building a JAR should run after Java compilation
- A task can thus **depend** on other tasks by calling **dependsOn**

```
1    task hello << {
2      println 'hello'
3    }
4
5    task goodbye {
6      dependsOn hello
7    }
8
9    goodbye << {
10     println 'goodbye'
11   }
```

If you run this script from the command line specifying the 'goodbye' task, the 'hello' task will run first.

A task can depend on more than one task. If so, you can pass a comma-separated list of the dependent task names to dependsOn().

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Plugins

- Gradle itself defines only a minimal number of tasks - you can add tasks using **plugins**
- Gradle defines several **standard** plugins (e.g. the **java** plugin) and there is an archive of third-party plugins
- To use a plugin, code an **apply plugin** line in the build script

```
1    apply plugin: 'java'
```

If you want to use a third-party plugin, you will need to download it, make it available on Gradle's CLASSPATH, then reference it in the "buildscript" section of the script. You can then use "apply plugin" to invoke it.

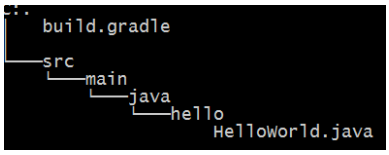
The documentation for the standard plugins is at: https://docs.gradle.org/current/userguide/standard_plugins.html.

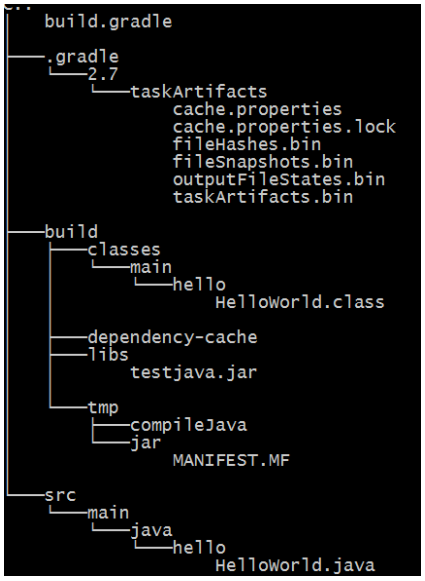
The plugin archive is at: https://plugins.gradle.org.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The java Plugin

- The **java** plugin provides tasks (e.g. **build**) to let you build and package Java applications
- By default, it expects a directory structure similar to Maven's





The pictures above show the 'before' and 'after' resulting from running the script on the last page with:

```
gradle build
```

If you want to use a different directory structure, the 'java' plugin makes that pretty easy (unlike Maven). You can provide the sourceSets property to let the plugin know about your structure. See https://docs.gradle.org/current/userguide/java_plugin.html for an example.

The plugin's task are dependent on each other; for example, the 'build' task depends on compileJava and so forth. See the docs for a picture of how the tasks depend on each other.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dependency Management

- Most projects use external libraries, e.g. the Hibernate ORM
- Gradle lets you specify **where** the dependencies come from using **repositories** and **which** dependencies using **dependencies**
- Libraries typically have their own dependencies, so Gradle supports **transitive dependency management** to fetch all required libraries

For repositories, Gradle understands Maven, Ivy and local file-systems.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dependency Management Example

```
1    apply plugin: 'java'
2
3    repositories {
4        mavenCentral()
5    }
6
7    dependencies {
8     compile group: 'org.hibernate', name: 'hibernate-core', \
9          version: '3.6.7.Final'
10    testCompile group: 'junit', name: 'junit', version: '4.+'
11   }
```


Here we specify that we need exactly Hibernate version 3.6.7.Final and any version of JUnit greater than or equal to version 4.0.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dependency Configurations

- You can group dependencies into **configurations**, which are a logical set of dependencies
- The **java** plugin defines a few configurations including **compile** (libraries needed to compile), **runtime** (libraries needed to run; includes compile libs) and **testCompile** (libraries need to test; includes compile libs)

Other plugins can define other configurations. For example, the 'war' plugin that builds a Web archive defines a 'providedRuntime' configuration that specifies the libraries to be added to the Web archive (WAR) file.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How Ant, Maven and Gradle compare to each other
- How to write and run simple Gradle build scripts

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum